The Volume-of-Tubes formula: Computational Methods and Statistical Applications

Catherine Loader
Department of Statistics
Case Western Reserve University
Cleveland, OH 44106

February 2, 2008

Abstract

The volume-of-tube formula was first introduced by Hotelling (1939), to solve significance of terms in nonlinear regression models. Since this pioneering paper, there has been significant work on extending the tube formula to more general settings, including multidimensional problems, and many new applications in statistical inference, including confidence bands in regression and smoothing models; applications to functional data analysis; testing in mixture models; and spatial scan analysis.

Implementation of the tube formula requires numerical evaluation of certain problem-specific geometric constants that appear in Hotelling's formula and its extensions. The purpose of this note is to describe a software library, `libtube`, that performs the calculations. A variety of illustrative examples are given.

Source code for the `libtube` library and examples can be downloaded from `http://www.herine.net/stat/libtube/`.

1 Introduction

The volume-of-tube problem can be stated rather simply. Given a curve (or manifold) $\mathcal{M}$ lying in n-dimensional Euclidean space, what is the volume of the set of all points lying within a radius r of the curve? In statistical applications, the spherical version of this problem often arises; the manifold lies on the surface of the unit sphere in n dimensions, and one wishes to compute the $(n-1)$-dimensional volume (or surface area) of the set of points lying within a distance r of the manifold.

The volume-of-tubes formula was formulated and solved by Hotelling (1939), motivated by application to significance testing in nonparametric regression. A companion paper, Weyl (1939), extended the results to higher dimensional manifolds; that is, when $\mathcal{M}$ is a surface, or more generally when $\mathcal{M}$ is a manifold of dimension $d \leq n$.

The main purpose of this article is to describe a set of routines written by the author to implement the volume-of-tube formula in statistical problems. In section 2 the tube formula (with boundary corrections) is described. The `libtube` software is described in Section 3. Applications to non-linear regression, simultaneous confidence bands and mixture modeling are described in Sections 4, 5 and 6 respectively.

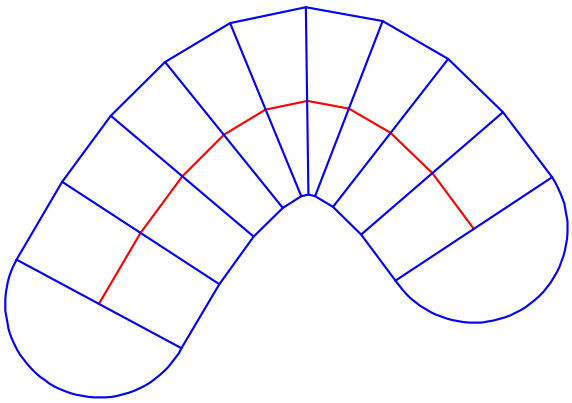

Figure 1: The manifold is represented by the red curve. The tubular neighborhood is approximated by trapezoids, plus the two end-point caps.

2 The Volume-of-Tubes Formula

The volume-of-tubes formula was first derived by Hotelling (1939). The result can be illustrated on the plane by reference to Figure 1. The manifold is represented by the red curve. The tubular neighborhood of a given radius r is approximated by trapezoids, plus the two end-point caps. Adding up the area of the trapezoids and letting the partition become increasingly fine shows that the area (or two-dimensional volume) of the tube is

$$\text{Length of Manifold} \times 2r + \pi r^2.$$

The $2r$ represents the cross-sectional area of the manifold, while πr^2 represents the area of the end-point caps. The result extends to manifolds embedded in n-dimensional space;

$$\text{Volume} = \kappa_0 V_{n-1} r^{n-1} + \frac{l_0}{2} V_n r^n.$$

Here κ_0 is the length of the manifold and l_0 is the number of end-points (often, $l_0 = 2$). The functions $\psi_0(r)$ are the cross-sectional area and volume of the end-point caps respectively, and $V_k = \pi^{k/2}/\Gamma(1 + k/2)$ is the volume of the k-dimensional unit sphere.

When the manifold lies on the unit sphere, the result is similar, but the cross-sectional area is replaced by a certain partial beta function. The result is

$$\text{Volume} = \frac{\kappa_0 A_n}{2\pi} P(B_{1,(n-2)/2} \geq w^2) + \frac{l_0 A_n}{4} P(B_{1/2,(n-1)/2} \geq w^2),$$

where $B_{a,b}$ denotes a random variable following a beta distribution with parameters a and b; $A_n = 2\pi^{n/2}/\Gamma(n/2)$ is the surface area of the unit sphere in R^n, and $w = 1 - r^2/2$.

Multidimensional Manifolds

Figure 2 shows a tube around a two-dimensional manifold. To compute the volume of the tubular neighborhood, one divides the tube into different pieces: a main piece, the half-cylinders around each edge, and wedges at each corner of the manifold. For higher dimensional manifolds, the ideas are similar, but there are more pieces to take care of.

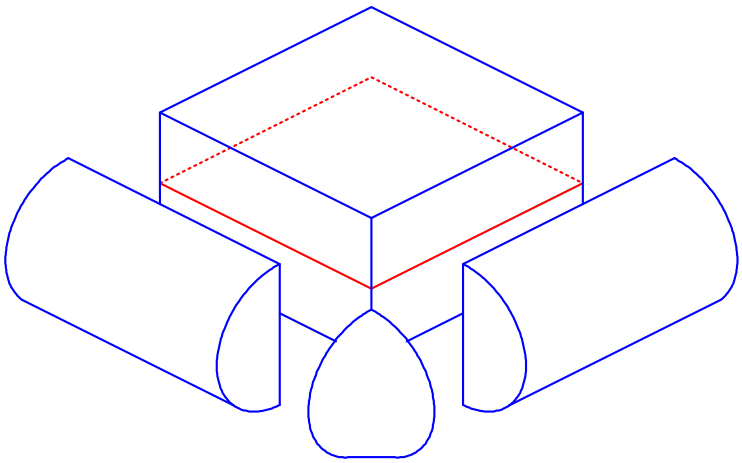

Figure 2: Tube around a two dimensional manifold. The manifold is shown in red, and the tube is divided into a main part, half-cylinders around the edges, and corner wedges.

A version of the tube formula, without boundary corrections, was first derived by Weyl (1939). Naiman (1990) provided boundary corrections. The result is a series with $d+1$ terms. The first four terms (for manifolds on the unit sphere) are

$$\begin{aligned}\text{Volume} \quad = \quad & \frac{\kappa_0 A_n}{A_{d+1}} P(B_{(d+1)/2,(n-d-1)/2} > w^2) \\ & + \frac{l_0 A_n}{2A_d} P(B_{d/2,(n-d)/2} > w^2) \\ & + \frac{\kappa_2 + l_1 + m_0}{2\pi} \frac{A_n}{A_{d-1}} P(B_{(d-1)/2,(n-d+1)/2} > w^2) \\ & + \frac{l_2 + m_1 + n_0}{4\pi} \frac{A_n}{A_{d-2}} P(B_{(d-2)/2,(n-d+2)/2} > w^2).\end{aligned}$$

The constants l_0, l_1 and l_2 arise from the corresponding series for the half-tubes around the boundaries of the manifold. l_0 is the $(d-1)$-dimensional volume of the boundaries (or the total length of the four edges in Figure 2). l_1 and l_2 are higher order terms representing boundary curvature.

m_0 and m_1 arise from the 'corner wedges' where two boundary faces meet. In figure 2, m_0 is the sum of the four wedge angles at each corner of the manifold.

n_0 arises for manifolds with $d \geq 3$, from the corners where three (or more) boundary faces meet.

2.1 Random Processes

In statistical applications, the fundamental use of the tube formula is to find (or at least approximate) the distribution of the maximum of certain random

processes. As a simple example, consider the process

$$Z(\lambda) = \langle T(\lambda), U \rangle$$

where $T(\lambda)$ is an R^n-valued vector function, and U is uniformly distributed over the unit sphere. Suppose that one is interested in finding

$$P(\sup_{\lambda} Z(\lambda) \geq w)$$

for some w.

The inner product exceeds w if, and only if, U is sufficiently close to $T(\lambda)$. Specifically,

$$\|T(\lambda) - U\|^2 = \|T(\lambda)\|^2 - 2\langle T(\lambda), U \rangle + \|U\|^2 = 2(1 - \langle T(\lambda), U \rangle).$$

Hence, $\|T(\lambda) - U\| \leq r$ if, and only if, $\langle T(\lambda), U \rangle \geq w$, where $r^2 = 2(1-w)$. The probability (2.1) is therefore simply

$$\frac{\text{Area of tube of radius } r \text{ around } \{T(\lambda)\}}{\text{Surface area of unit sphere in } R^n}.$$

In many statistical applications, one is interested in the distribution of the maximum of a Gaussian process,

$$Z(\lambda) = \langle T(\lambda), \epsilon \rangle$$

where ϵ follows the standard multivariate normal distribution. To reduce this to the uniform process, one needs to condition on the length of the ϵ vector, and integrate over the conditional distribution; see Sun and Loader (1994). The final result, up to fourth order, is

$$\begin{aligned} P(\sup Z(\lambda) \geq c) \quad \approx \quad & \frac{\kappa_0}{A_{d+1}} P(\chi^2_{(d+1)/2} \geq c^2) \\ & + \frac{l_0}{2A_d} P(\chi^2_{d/2} > c^2) \\ & + \frac{\kappa_2 + l_1 + m_0}{2\pi A_{d-1}} P(\chi^2_{(d-1)/2} > c^2) \\ & + \frac{l_2 + m_1 + n_0}{4\pi A_{d-2}} P(\chi^2_{(d-2)/2} > c^2), \end{aligned}$$

where χ^2_k denotes a chi-square random variable with k degrees of freedom.

3 The `libtube` Library

The main computational problem in implementing results based on the tube formula is evaluation of the constants κ_0, κ_2, l_0 e.t.c. The `libtube` library implements the tube library up to fourth order terms. To use the library, one must first write a 'manifold function' defining the problem. `libtube` takes the manifold function as input, and uses numerical integration methods to compute the constants.

The library can be downloaded from `http://www.herine.net/stat/libtube`. The library is written in C, and can be compiled on Linux systems using

```
% make
% make install
```

to install the libraries in `/usr/local/lib`.

```
% cc -o nlreg nlreg.c -ltube -lmut -lm
```

The library (and the examples given in this paper) have been written and tested using the Gnu C compiler available in most Linux distributions. The C code should be compatible with most other compilers and operating systems.

3.1 Manifold Functions

Suppose the manifold is defined by a vector function $T(x)$ mapping a d-dimensional domain $\mathcal{X}$ to the manifold $\mathcal{M}$ in n-dimensional space. The constants in the tube formula can be computed from $T(x)$ and its derivatives, so in it's simplest form, the manifold function simply computes these. In statistical applications, one usually doesn't get $T(x)$ naturally, but rather one gets a vector $l(x)$ such that $T(x) = l(x)/\|l(x)\|$ (see the regression examples in Sections 4 and 5). The manifold function can instead provide $l(x)$ and its derivatives.

In still other examples, one doesn't even obtain $l(x)$ directly, but instead obtains a covariance function $\sigma(x, x') = \langle l(x), l(x')\rangle$ (see the mixture example, Section 6). Since the distance between any two points on the manifold is given by

$$\|l(x) - l(x')\|^2 = \sigma(x, x) + \sigma(x', x') - 2\sigma(x, x'),$$

knowledge of the covariance function determines $l(x)$ up to an orthogonal transformation. The manifold function can provide $\sigma(x, x')$ and its derivatives.

The precise form of the manifold functions is illustrated by the examples. After writing the manifold function, the most useful functions in `libtube` are:

- `tube_contstants()`, to numerically evaluate κ_0 and the other constants appearing in (??).
- `tailp()` and `critval()`, which compute tail probabilities corresponding to a specified cut-off, and critical values corresponding to a specified significance level.

Calling sequence for `tube_constants()`.

The function to compute the constants is

```
int tube_constants(f, d, n, ev, mg, fl, kap, wk, deb, uc);
int (*f)();
int d, n, ev, mg;
double *fl, *kap, *wk;
int deb, uc;
```

The arguments to this function are:

- `f` The manifold function to compute $l(x)$ and its derivatives.
- `d` The dimension of the manifold.
- `m` The maximum length of the $l(x)$ vectors. The argument provided is only used to allocate work space; the actual length of $l(x)$ is returned by the manifold function.
- `ev` Integration type. For rectangular domains, `ISIMPSON` is the most useful.
- `mg` Integer vector, giving the number of partitions to use in each dimension of the numerical integration rules.
- `fl` Integration limits. A numeric vector with length $2d$. The first d components give lower limits for each variable; the remaining d components give upper limits.
- `kap` is the vector through which the computed constants are returned. It should be allocated with at least $\min(d + 1, 4)$ terms. The values returned are $\kappa_0, l_0/2, (\kappa_2 + l_1 + m_0)/(2\pi)$ and $(l_2 + m_1 + n_0)/(4\pi)$.
- `wk` is a workspace vector. If `wk=NULL`, the required workspace will be allocated and freed within the `tube_constants()` function. To pre-allocate the space, the required length can be found by calling `k0_reqd(d,m)`.

- `terms` Number of terms to compute, from 1 to 4.
- `uc` An indicator variable indicating whether the manifold function computes the weight vectors `uc=0` or covariance derivative matrix `uc=1`.

Calling sequence for `tailp()` *and* `critval()`.

Tail probabilities are computed using the function `tailp`:

```
double tailp(c,k0,m,d,s,n,process)
double c, *k0, n;
int m, d, s, process;
```

Critical values corresponding to a specified tail probability are computed using the `critval` function:

```
double critval(alpha,k0,m,d,s,n,process)
double *k0, al, n;
int m, d, it, s;
```

These arguments represent:

- `c` Cut-off value for `tailp()`.
- `alpha` tail probability for `critval()`.
- `k0` is the vector of constants computed by the `tube_constants()` function.
- `m` is the number of terms in `k0`. This is the returned value of `tube_constants()`, and is equal to $\min(d+1, 4)$.
- `d` is the dimension of the manifold.
- `c` critical value (`tailp()` only).
- `s` Either `ONE_SIDED` or `TWO_SIDED`.
- `n` For the t-process, the residual degrees of freedom used to estimate σ. For the uniform process, the dimension n. Ignored for the Gaussian process. Beware that n must have type double.
- `process` Either `GAUSS` (when ϵ is multivariate Gaussian); `TPROC` (Gaussian process with estimated variance); or `UNIF` (when ϵ is uniform on the unit sphere).

3.2 Writing a Manifold Function with vectors

The manifold function computes the vector $l(x)$ and its derivatives. The basic form of the function is

```
int mymf(x,l,reqd)
double *x, *l;
int reqd;
{ /* function body goes here */
}
```

The `x` argument is a point in the input space; `l` is a vector to be filled in by the manifold function. The final argument, `reqd`, is an integer indicating what the library requires from the manifold function. If `reqd=0`, only the vector $l(x)$ is required. If `reqd=1`, then both $l(x)$ and $l'(x)$ (or all the first-order partial derivative vectors of $l(x)$) are required. If `reqd=2`, then additionally the second-order partial derivative vectors are required.

In statistical applications, the manifold function will generally require a data vector, sample size n, dimension d and variables other than x in order to perform its calculations. These variables should be assigned to global variables so that they are accessible in the manifold function.

The results of the computations are returned through the `l` vector. The vector $l(x)$ is placed in the first n elements. The first-order derivatives are placed in the next $n \times d$ elements. The second-order derivatives are placed in the next $n \times d \times d$ elements.

The function should return n, the length of the vector $l(x)$. Generally, this should be equal to the n value provided in the `tube_constants()` call; it should never be larger. It can be less. An example where it may be less is for a kernel regression with compactly supported kernel; only the non-zero elements of $l(x)$ need be retained.

3.3 Writing a Manifold Function with a covariance function.

The structure of a manifold function based on the covariance is identical to the vector case; it differs in what is computed.

Given a covariance function $\sigma(x, x')$, the manifold function needs to compute (in the one-dimensional case),

$$\begin{pmatrix} \sigma(x,x') & \frac{\partial\sigma(x,x')}{\partial x'} & \frac{\partial^2\sigma(x,x')}{\partial x'^2} \\ \frac{\partial\sigma(x,x')}{\partial x} & \frac{\partial^2\sigma(x,x')}{\partial x\partial x'} & \frac{\partial^3\sigma(x,x')}{\partial x\partial x'^2} \\ \frac{\partial^2\sigma(x,x')}{\partial x^2} & \frac{\partial^3\sigma(x,x')}{\partial x^2\partial x'} & \frac{\partial^4\sigma(x,x')}{\partial x^2\partial x'^2} \end{pmatrix},$$

evaluated at $x' = x$. Again, the matrix is stored in the vector `l`, with the columns stacked atop each other.

In higher dimensions, the required matrix is most easily written in terms of differential operators. The required $(1 + d + d^2) \times (1 + d + d^2)$ matrix is

$$\begin{pmatrix} I \\ D_{x_1} \\ \vdots \\ D_{x_d} \\ D_{x_1,x_1} \\ \vdots \\ D_{x_d,x_d} \end{pmatrix} \sigma(x,x') \begin{pmatrix} I & D_{x'_1} & \dots & D_{x'_d} & D_{x'_1,x'_1} & \dots & D_{x'_d,x'_d} \end{pmatrix}$$

where D represents the partial derivative operator with respect to the subscripted variables.

Another view is as follows. If $\mathbf{L}$ is the matrix computed by a manifold function with vectors, then $\mathbf{L}^T\mathbf{L}$ is the matrix computed by a manifold function with a covariance function.

4 Example: Testing in Nonlinear Regression

This was the motivating example for Hotelling (1939), and was developed in much more detail by Knowles and Siegmund (1989). Suppose one has data $(x_i, Y_i), i = 1, \dots, n$, and a nonlinear regression model, such as

$$Y_i = \alpha e^{\gamma x_i} + \epsilon_i. \tag{1}$$

The important feature of this model is that the parameter α enters the model linearly, while γ enters nonlinearly. Assume that the errors are independent $N(0, \sigma^2)$.

Consider the problem of testing $H_0 : \alpha = 0$ vs $H_1 : \alpha \neq 0$. It can be shown that the log-likelihood ratio test statistic is equivalent to

$$L = \frac{\min_{\alpha,\gamma} \|Y - al(\gamma)\|^2}{\|Y\|^2} \tag{2}$$

where $l(\gamma)^T = (e^{\gamma x_1}, \ldots, e^{\gamma x_n})$.

In classical statistical theory, log-likeliood ratio statistics often have asymptotic χ^2 distributions. However, this is not the case for the statistic (2). One way to see this is to recall that proofs of the χ^2 results are based on a quadratic expansion of the statistic under the null parameters. For the present problem this would require an expansion around $(0, \gamma_0)$ where γ_0 is 'the' null value of γ. Unfortunately this is undefined: when $\alpha = 0$, the parameter γ does not appear in (1); it is not identifiable!

For fixed γ, minimizing over a is a linear least-sqaures problem. It follows that

$$L = 1 - \sup_{\gamma} \left\langle \frac{l(\gamma)}{\|l(\gamma)\|}, \frac{Y}{\|Y\|} \right\rangle^2 .$$

The null hypothesis H_0 is rejected if $L \leq 1 - w^2$ for some $w > 0$, or equivalently, if

$$\sup_{\gamma} \left| \left\langle \frac{l(\gamma)}{\|l(\gamma)\|}, \frac{Y}{\|Y\|} \right\rangle \right| \geq w.$$

The constant w must be chosen to obtain a specified significance level. That is, we need to be able to evaluate probabilities of the form

$$P(\sup_{\gamma} | \langle T(\gamma), U \rangle | \geq w) \tag{3}$$

where $T(\gamma) = l(\gamma)/\|l(\gamma)\|$ defines a curve on the unit sphere, and $U = Y/\|Y\|$ is (under $H_0 : \alpha = 0$) uniformly distributed on the surface of the sphere.

4.1 Non-linear Regression: Implementation

Code implementing the tube formla for the non-linear regression problem is shown below. The program consists of the manifold function `regmf()`, and the main routine `main()` that reads in the data and computes the tube constants. Note that the data vectors and sample size are stored as global variables, so that they can be accessed within the manifold function.

The manifold function computes the components of $l(\gamma)$; $l_i(\gamma) = e^{\gamma x_i}$, and of $l'(\gamma)$, $l'_i(\gamma) = \gamma e^{\gamma x_i}$. These vectors are stored end-to-end in the `l` argument.

```
#include <stdio.h>
#include <math.h>
#include <tube.h>
#define MAXN 1000

double x[MAXN], y[MAXN];
int n;

int regmf(gam,l,reqd)
double *gam, *l;
int reqd;
{ int i;
  double *l1;
  l1 = &l[n];
  for (i=0; i<n; i++)
  { l[i] = exp(gam[0]*x[i]);
```

```
    l1[i] = x[i]*exp(gam[0]*x[i]);
  }
  return(n);
}

int main()
{ FILE *infile;
  char filename[100];
  int i, mg;
  double gamlimits[2], kappa[4];
  printf("Data filename ? "); scanf("%s",filename);
  printf("n = ? "); scanf("%d",&n);
  infile = fopen(filename,"r");
  for (i=0; i<n; i++) fscanf(infile,"%lf%lf",&x[i],&y[i]);
  gamlimits[0] = -2.0;
  gamlimits[1] = 2.0;
  mg = 100;

  tube_constants(regmf,1,n,ISIMPSON,&mg,gamlimits,kappa,NULL,0,0);
  printf("%8.5f %8.5f\n",kappa[0],kappa[1]);
}
```

5 Example: Simultaneous Confidence Bands

Application of the tube formula to find simultaneous confidence bands for regression models has been studied in Naiman (1987), Sun and Loader (1994) among others. Consider again regression data, but now suppose that the model is

$$Y_i = a_0 + a_1 x_i + a_2 x_i^2 + \epsilon_i = \mu(x_i) + \epsilon_i$$

(although we formulate the problem for quadratic regression, extension to other linear models is straightforward). The goal is to find confidence bands

$$\hat{\mu}(x) \pm c\sqrt{\mathrm{var}(\hat{\mu}(x))}$$

with simultaneous coverage over some nice domain $\mathcal{X}$:

$$P(|\hat{\mu}(x) - \mu(x)| \le c\sigma\|l(x)\| \text{ for all } x \in \mathcal{X}) = 1 - \alpha. \tag{4}$$

The least-squares estimates of the parameters are

$$\begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \hat{a}_2 \end{pmatrix} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T Y$$

where $\mathbf{X}$ is the design matrix. For fixed x, $\mu(x)$ is estimated by

$$\hat{\mu}(x) = \hat{a}_0 + \hat{a}_1 x + \hat{a}_2 x^2 = (\,1 \quad x \quad x^2\,)\,(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T Y = \langle l(x), Y\rangle\,,$$

and the variance of the estimate is $\mathrm{var}(\hat{\mu}(x)) = \sigma^2\|l(x)\|$.

Now, $\hat{\mu}(x) - \mu(x) = \langle l(x), \epsilon\rangle$, and the probability (4) is equivalent to

$$\alpha = P\left(\sup_x |\left\langle \frac{l(x)}{\|l(x)\|}, \epsilon \right\rangle| > c\right).$$

This problem can be solved using the Gaussian process variant of the tube problem.

Suppose $\mathbf{X} = \mathbf{QR}$ is the QR-decomposition of the design matrix. Then $l(x)$ lies in the column space of $\mathbf{Q}$ for all x, and so

$$Z(\gamma) = \left\langle \frac{\mathbf{Q}^T l(x)}{\|\mathbf{Q}^T l(x)\|}, \mathbf{Q}^T \epsilon \right\rangle,$$

so it suffices to work with the vector $l^*(x) = \mathbf{Q}^T l(x) = (\mathbf{R}^T)^{-1} f(x)$ where $f(x)$ is a vector of the polynomial basis functions. The derivatives are easily found;

$$\frac{d}{dx} l^*(x) = (\mathbf{R}^T)^{-1} \frac{d}{dx} f(x)$$

and so on.

5.1 Simultaneous Confidence Bands: Implementation

Code for the quadratic regression computations, in an arbitrary number of dimensions, is shown below. The functions `quad()`, `quadi()` and `quadij()` compute the quadratic basis functions $f(x)$; first-order partial derivatives and second-order partial derivatives respectively. The manifold function is `quadmf()`. The `main()` function reads the data from a file; computes the design matrix and its QR-decomposition; and then calls the `tube_constants()` function (The QR functions, `qr()` and `qrtinvx()`, as well as `transpose()`, are part of the `mut` library).

When the program is run, the user is prompted for a data file (containing a matrix of the predictor variables); data dimension (n and d), and limits for the confidence band computation.

The tube constants are computed, then the critical value c for 95% confidence bands. Note that the final argument to `critval` is the residual degrees of freedom used to estimate σ; Sun and Loader (1994) give the modification of (**??**) for this case.

```
#include <stdio.h>
#include <tube.h>
#include <mutil.h>

int dim, n, p;
double *X;

void quad(x,f)
double *x, *f;
{ int i, j, k;
  k = 0;
  f[k++] = 1.0;
  for (i=0; i<dim; i++) f[k++] = x[i];
  for (i=0; i<dim; i++)
    for (j=i; j<dim; j++)
      f[k++] = x[i]*x[j];
}

void quadi(x,f,i0)
double *x, *f;
int i0;
{ int i, j, k;
  k = 0;
  f[k++] = 0.0;
```

```
  for (i=0; i<dim; i++) f[k++] = (i==i0);
  for (i=0; i<dim; i++)
    for (j=i; j<dim; j++)
      f[k++] = (i==i0)*x[j] + (j==i0)*x[i];
}

void quadij(x,f,i0,j0)
double *x, *f;
int i0, j0;
{ int i, j, k;
  k = 0;
  f[k++] = 0.0;
  for (i=0; i<dim; i++) f[k++] = 0.0;
  for (i=0; i<dim; i++)
    for (j=i; j<dim; j++)
      f[k++] = ((i==i0) & (j==j0)) + ((i==j0) & (j==i0));
}

int quadmf(x,l,reqd)
double *x, *l;
int reqd;
{ int i, j, k;
  k = 0;
  quad(x,l);
  qrtinvx(X,l,n,p);
  k++;
  for (i=0; i<dim; i++)
  { quadi(x,&l[k*p],i);
    qrtinvx(X,&l[k*p],n,p);
    k++;
  }
  for (i=0; i<dim; i++)
    for (j=0; j<dim; j++)
    { quadij(x,&l[k*p],i,j);
      qrtinvx(X,&l[k*p],n,p);
      k++;
    }
  return(p);
}

int main()
{ FILE *infile;
  char filename[100];
  int i, j, mg[100];
  double xlim[100], kappa[4], datarow[100];
  printf("Data filename ? "); scanf("%s",filename);
  printf("n = ? "); scanf("%d",&n);
  printf("dim = ? "); scanf("%d",&dim);
  infile = fopen(filename,"r");
  if (infile==NULL)
  { printf("Error: can't read input file\n");
    return(0);
  }
  p = 1 + dim + dim*(dim+1)/2;
  X = (double *)calloc(n*p,sizeof(double));
```

```
  for (i=0; i<n; i++)
  { for (j=0; j<dim; j++)
      fscanf(infile,"%lf",&datarow[j]);
    quad(datarow,&X[i*p]);
  }
  transpose(X,n,p);
  qr(X,n,p,NULL);
  for (i=0; i<dim; i++) mg[i] = 20;
  xlim[0] = -2; xlim[1] = -2;
  xlim[2] = 2; xlim[3] = 2;
  tube_constants(quadmf,dim,p,ISIMPSON,mg,xlim,kappa,NULL,3,0);
  printf("kappa: %8.5f %8.5f %8.5f %8.5f\n",kappa[0],kappa[1],kappa[2],kappa[3]);
}
```

6 Example: Mixture Models

Suppose $X_1, \ldots, X_n$ are an i.i.d. sample from a density

$$f_{\alpha,\lambda}(x) = (1-\alpha)f_0(x) + \alpha\phi(x,\lambda)$$

where α and λ are unknown parameters, with $0 \le \alpha \le 1$. The object is to test $H_0 : \alpha = 0$ vs $H_1 : \alpha > 0$. This is a simple example of mixture testing: under H_0, the single component $f_0(x)$ describes the data, while under H_1, the two components are required.

Consider the normalized score process proposed by Pilla and Loader (2003). The score process is

$$S(\lambda) = \sum_{i=1}^{n} \frac{\phi(X_i,\lambda)}{f_0(X_i)} - 1.$$

Under the null hypothesis, this has mean 0 and covariance function $n\sigma(\lambda,\lambda^\dagger)$, where

$$\sigma(\lambda,\lambda^\dagger) = \int \frac{\phi(x,\lambda)\phi(x,\lambda^\dagger)}{f_0(x)}dx - 1.$$

The normalized score process is $S^*(\lambda) = S(\lambda)/\sqrt{n\sigma(\lambda,\lambda)}$. This asymptotically behaves like a Gaussian process $Z(\lambda)$, with mean 0 under H_0, and a nonzero mean under H_1. The maximum of the normalized score process serves as the test statistic.

Since an explicit vector representation of $Z(\lambda)$ is not readily available, the manifold function (`mixmf` in the code below) must be written using the covariance function and its partial derivatives.

There is one additional difficulty. The normalized score process has a singularity at $\mu = 0$. For this reason, the manifold function works with Taylor series expansions of the covariance in this reason. Also, the singularity results in a discontinuity in $S^*(\lambda)$, and $l_0 = 4$.

The main routine in the program below sets limits for μ, calls the `tube_constants()` function, and computes the 5% critical value.

```
kappa0 =  5.27449
  l0/2 =  2.00000
Level 0.05 critical value =  2.49455
```

```
#include <stdio.h>
#include <math.h>
#include <tube.h>
#define MAXN 1000
```

```
double x[MAXN], y[MAXN];
int n;

int mixmf(mu,l,reqd)
double *mu, *l;
int reqd;
{ double emm, mm;

  if (fabs(mu[0]) < 0.01)
  { mm = mu[0]*mu[0];
    l[0] = 1 + mm/2*(1 + mm/3*(1 + mm/4*(1 + mm/5)));
    l[1] = 0.5*(1 + mm/3*(2 + mm/4*(3 + mm/5*(4 + 5/6*mm))));
    l[1] = l[2] = mu[0]*l[1];
    l[3] = 0.5*(1 + mm/3*(4 + mm/4*(9 + mm/5*(16+25/6*mm))));
  } else
  { emm = exp(mu[0]*mu[0]);
    l[0] = emm-1;
    l[1] = l[2] = mu[0]*emm;
    l[3] = emm*(1+mu[0]*mu[0]);
  }
  return(2);
}

int main()
{ int i, mg, t;
  double mulimits[2], kappa[4];
  mulimits[0] = -3.0;
  mulimits[1] = 3.0;
  mg = 200;

  t = tube_constants(mixmf,1,n,ISIMPSON,&mg,mulimits,kappa,NULL,2,1);
  /* modify kappa[1] = l0/2 for the singularity */
  kappa[1] += 1.0;

  printf("kappa0 = %8.5f\n",kappa[0]);
  printf("  l0/2 = %8.5f\n",kappa[1]);
  printf("Level 0.05 critical value = %8.5f\n",critval(kappa,t,1,0.05,10,1,0.0));
}
```

References

Hotelling, H. (1939). Tubes and spheres in n-spaces, and a class of statistical problems. *American Journal of Mathematics*, 61:440–460.

Knowles, M. and Siegmund, D. (1989). On Hotelling's geometric approach to testing for a nonlinear parameter in regression. *International Statistical Review*, 57:205–220.

Naiman, D. Q. (1987). Simultaneous confidence bounds in multiple regression using predictor variable constraints. *Journal of the American Statistical Association*, 82:214–219.

Naiman, D. Q. (1990). On volumes of tubular neighborhoods of spherical polyhedra and statistical inference. *The Annals of Statistics*, 18:685–716.

Pilla, R. S. and Loader, C. (2003). The volume-of-tube formula: Perturbation tests, mixture models and scan statistics. Unpublished Manuscript.

Sun, J. and Loader, C. (1994). Simultaneous confidence bands for linear regression and smoothing. *The Annals of Statistics*, 22:1328–1345.

Weyl, H. (1939). On the volume of tubes. *American Journal of Mathematics*, 61:461–472.